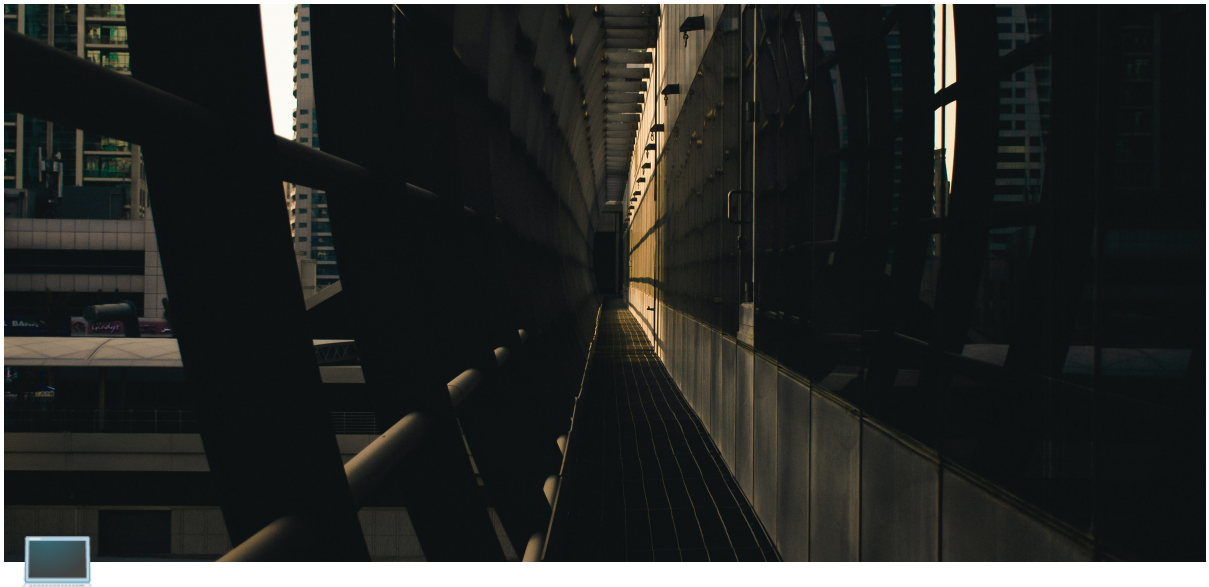


Deep-learning based channel decoding of short packets

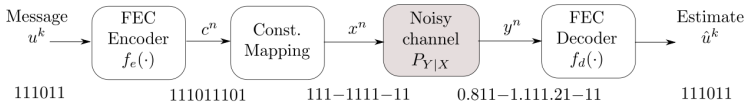
Léo Innocenzi & Jean Debouge-Boyer

Introduction

This presentation is axed on the reproducibility of a research paper concerning a deep-learning method for decoding short packets when Gaussian noise is present in a signal. The paper states that Machine Learning (ML), and particularly Deep Learning (DL) methods can be applied to learn decoding algorithms. This would have the benefit of cutting down processing power costs when receiving a lot of signal, and the fact that it can be applied to short-packets would be beneficial in a large array of technologies leveraging the Internet of Things. We will attempt to reproduce the technics used in the paper to make a Neural Network learn a decoding algorithm from a polar code, explain them and try to analyse the results we got.

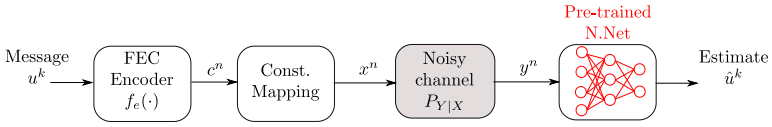
The algorithm

The algorithm we would want to make a neural net learn in called the Maximum A Posteriori decoding algorithm. It is an algorithm that repetitively compares the noisy signal to all possible information bit known to exist in the codebook of the encoded signal. For an 8-bit signal encoded with a polar code to a 16-bit transmitted signal, it will compare it to 2^8 possible codewords in a '*dictionary*', and choose the one that is has the least distance to the incoming signal.



Example of an end to end channel with encoding and decoding.

Although this method has been proven to work, it is computationally expensive and requires both electrical power and processing power to work well and fast. The type of architecture we would like to create would resemble the following picture.



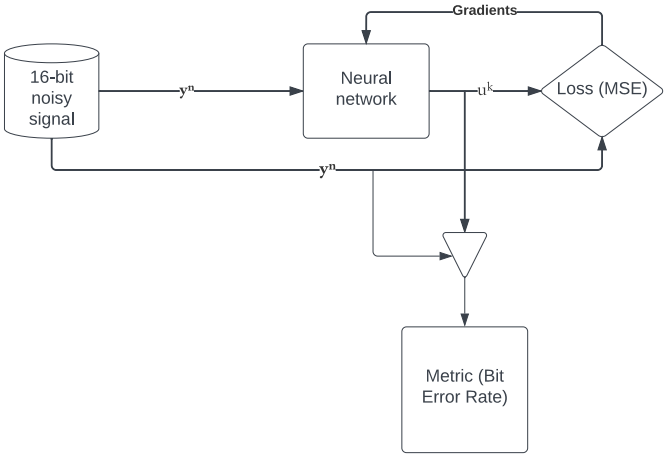
Example of an end to end channel with encoding and neural-net decoding.

As neural networks are defined only when they are trained, and composed only of weights and nodes values, their computation needs are relatively small compared to what it used today, and they would, in theory, speed up a lot the decoding of a noisy signal.

The neural network

A brief word about how neural networks learn

Neural networks learn with an error-correcting loop, ingesting data in entry, and trying its best to match the wanted result by calculating an error (loss), and maximising the metric, which is in our case the Bit Error Rate of our transmitted signal by reference to the original signal.



This loop repeats for every batch of data it sees, and readapts the weights and biases inside the neural network to provide a better estimation for the next training example. We are then minimising the loss while approximating a function, in our case the decoding of noisy packets. The metric is here for us the understand what the neural network is doing, and if it is learning what we asked.

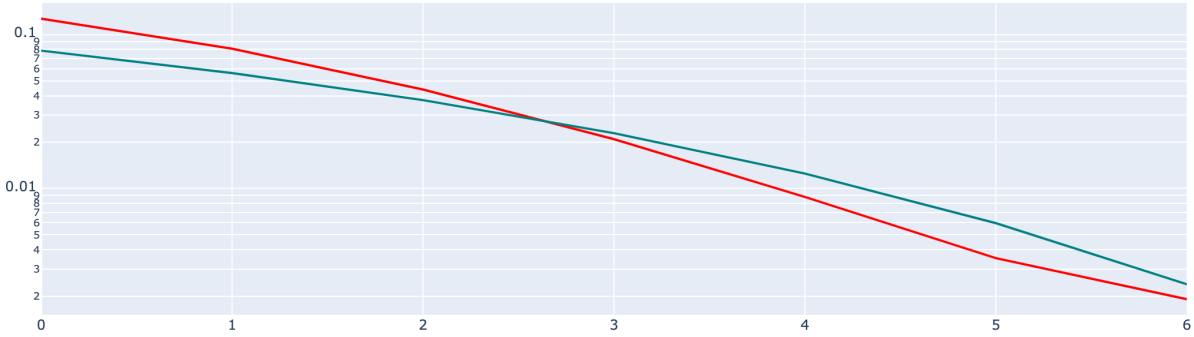
Requirements

Data

As said in the paper, we are limited by 2^8 information bits as training data, but by adding Gaussian white noise randomly in every loop, we can virtually create infinite amount of data. This is very convenient, as it will prevent any over-fitting or under-fitting to happen during training. Labelling data is also very easy, as we are talking about bits, we don't have to bother about any labelling part.

We started by creating an codebook in 8-bits containing every 2^k number and coded it by multiplying the codebook by the matrix `G`. This will be the only data fed into the neural net during training.

Testing will be different though, as we have to test random unseen data formats. We created a million bit-long dataset which we coded through the same polar code we used for the training data, and simultaneously computed the Bit Error Rate of the data through a classical MAP channel as well as with the neural network. This way, we can compare the time and accuracy of both methods. For reference, we have the following curves that compares the Theoretical BER with the MAP algorithm on a 10^5 long bitstream for a $\frac{E_b}{N_0}$ value ranging from 0 to 6.



Neural Network

The neural network we built is similar to the one described in the paper, with a few little different things we wanted to try. First, here is the neural network architecture we reproduced.

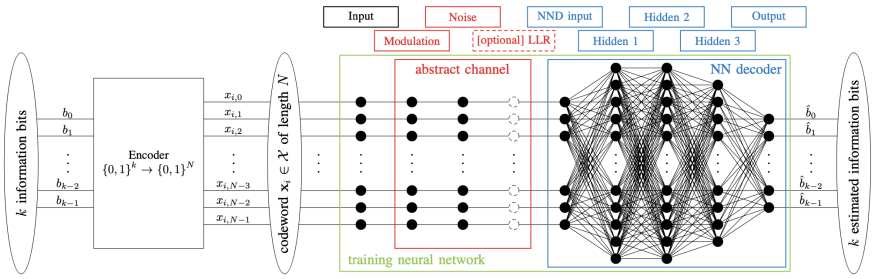


Fig. 1: Deep learning setup for channel coding.

The encoder is the `G` matrix we introduced earlier, and the structure of the neural network is the same, without the optional `LLR` layer. Two things were added during training :

1. We added a **learning-rate scheduler**, which decreases the learning rate when the loss encounters a plateau, to try to speed up the learning process and reduce the number of epochs needed.
2. We added a `BatchNormalization` layer just before the sigmoid output of the neural network (i.e. the just before the output layer). Batch Normalisation has a known effect to reduce the number of needed epochs to perform the learn the same. It is kind of a booster for the neural network to learn faster and retain the same informations. Of course, we will compare with and without to see if it has a real effect in our case.

The neural network is trained on 256 codewords of 16 bits corresponding to the 8-bits codewords of 2^k values. We train the neural network with a $\frac{E_b}{N_0} = 1.0$.

Results

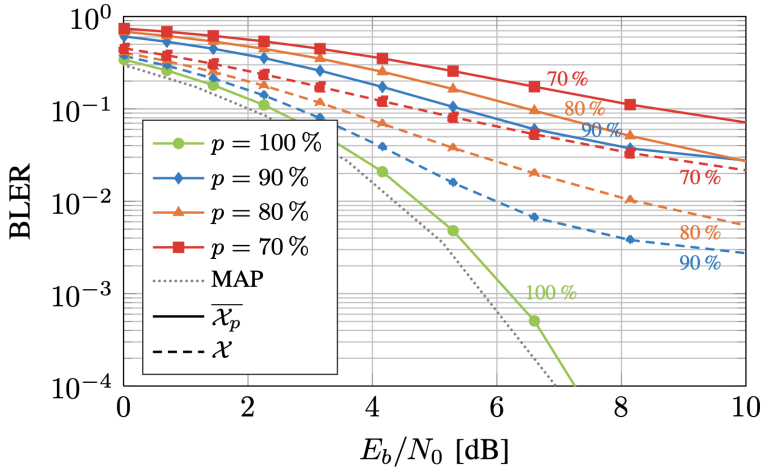
Without Batch Normalisation

Parameters

The results we will be discussing in this section outcome from a neural network trained with the following parameters :

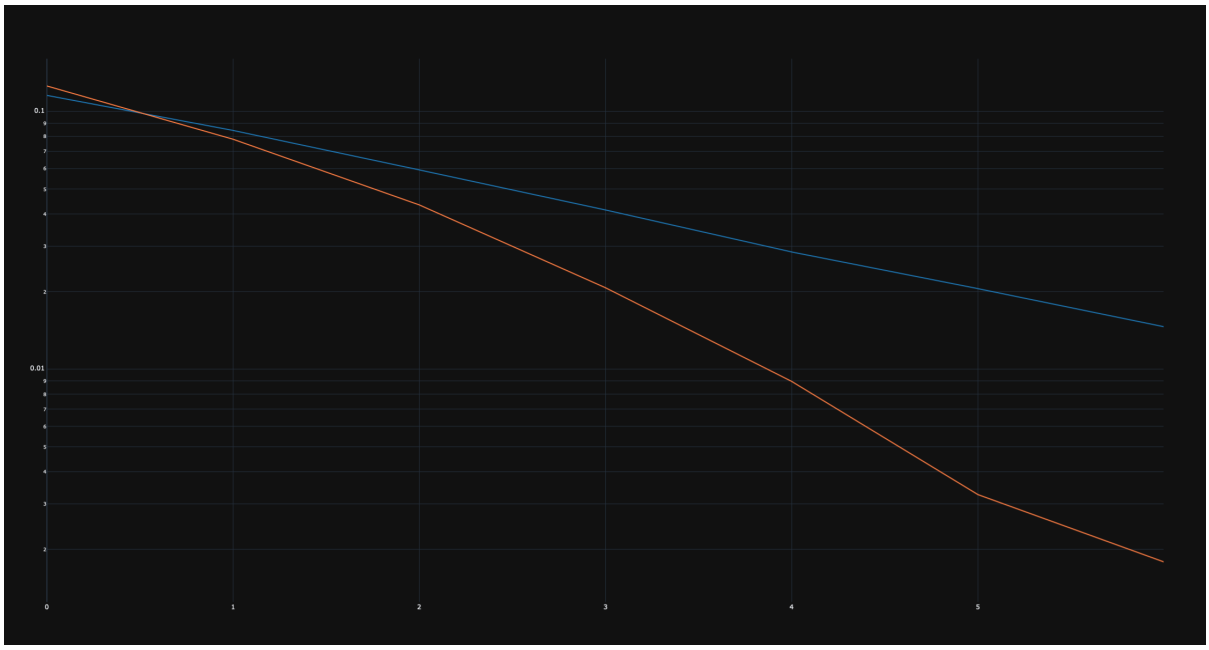
Parameter	Value
Number of epochs	**262 144 (2^18)**
Architecture	128-64-32
Learning rate	0.003 at start and scheduled down on plateaus
Batch size	255
Training proportion	100% of the 8-bit codebook used during training

Taking into account the parameters we used in this method, we would have to have something like the green curve in the following diagram, in reference to the dotted line showing the MAP algorithm's performance.



(a) 16 bit-length Polar Code ($r = 0.5$)

We got the following results, with the MAP performance in red and our Neural Network in blue, with a $\frac{E_b}{N_0}$ value ranging from 0 to 6 and a bitstream of 200 000 bits long for test.



From these results, we can definitely say that the algorithm has learned a way of decoding the noisy signals it received, but we can also see that it doesn't exactly matches the reference from the paper we tried to reproduce.

There may be a lot of different causes for this difference in performance, one may be the way we tried to create our neural network from scratch with custom Tensorflow classes, which may be missing some of the built-in features Tensorflow offers off the shelf. It may also be because of the processing power of the laptop we trained this on, which took approximately 1 hour to fully train with tensorflow functions optimisations.

That said, we can note that the neural network performed much faster decoding that the MAP algorithm. In the following table, 200 000 bits are decoded by the MAP Algorithm and the NN.

Eb/No	1	2	3	4
MAP result (BER)	0.07798	0.043375	0.02069	0.00896
MAP time (s)	47.3	47.8	48.8	47.5
NN result (BER)	0.08438	0.0592	0.0414	0.0284
NN time (s)	0.0128	0.0167	0.01317	0.01281

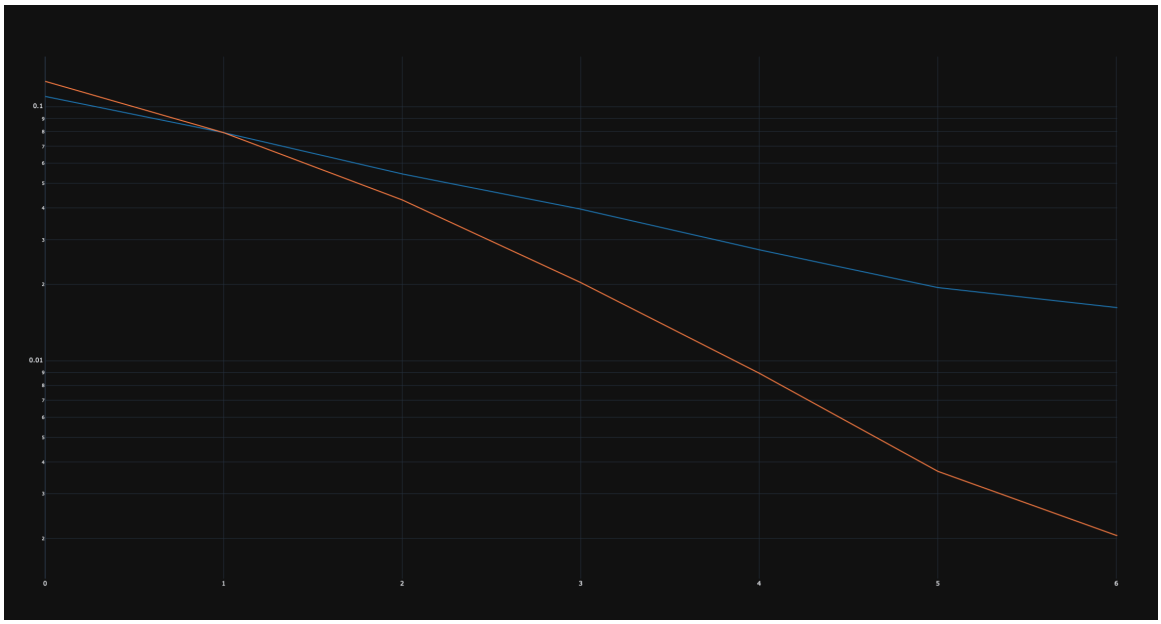
With Batch Normalisation

Parameters

The results we will be discussing in this section outcome from a neural network trained with the following parameters :

Parameter	Value
Number of epochs	**16 348 (2^14)**
Architecture	128-64-32
Learning rate	0.003 at start and scheduled down on plateaus
Batch size	255
Training proportion	100% of the 8-bit codebook used during training

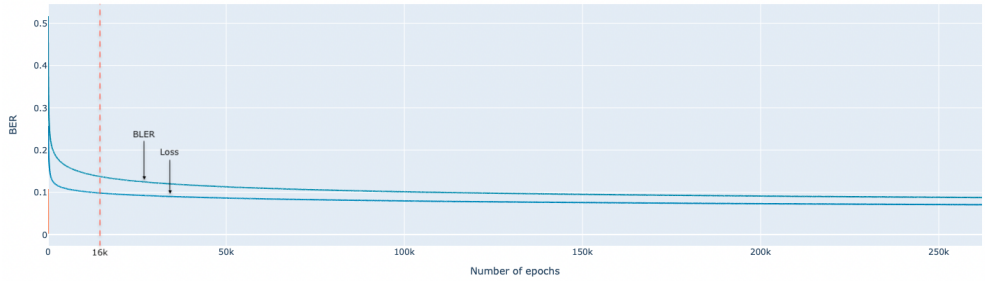
Again, we got the following results, with the MAP performance in red and our Neural Network in blue, with a $\frac{E_b}{N_0}$ value ranging from 0 to 6 and a bitstream of 200 000 bits long for test. The only difference in the number of epochs and the addition of the `BatchNormalization` layer before the output of the neural network.



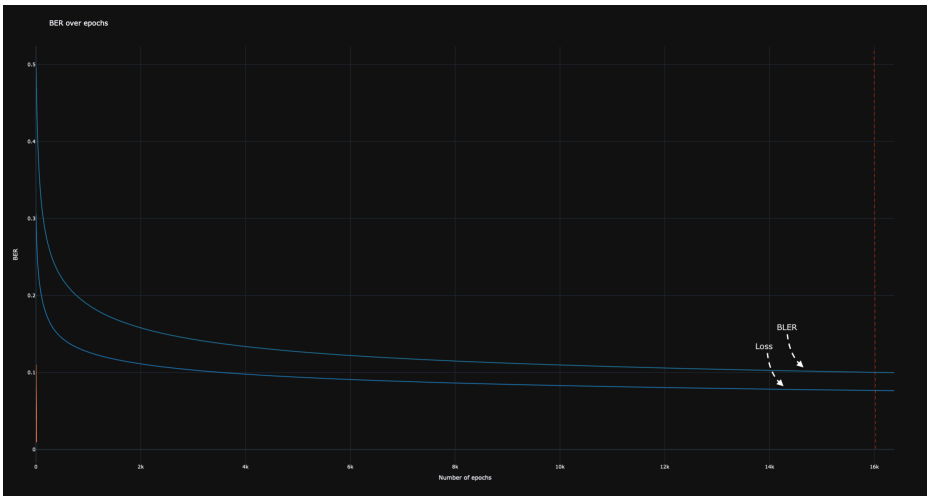
From this, we can conclude that the neural network learned the same way as it did without the Batch Normalisation adding. The results are sensibly the same, but the **number of epochs is only 6.25%** of what was used in the original paper. The addition of this layer is thus greatly efficient for this method. This can mean that retraining neural networks for decoding purposes, if coding is changed for any reason, can be done much faster and efficiently.

For comparaison, we have evolutions of the loss and Block Error Rate (Block being the BER over a batch) throughout the training process :

Without Batch Normalisation :



With Batch Normalisation :



Conclusion

In this work, we tried recreating a DL method for decoding short-packets of noisy data. We think we partially succeeded to implement the work from the paper because it did not have the desired accuracy during testing we thought it could have, but we explored more on a problem that slowed us down during the implementation of the paper. Indeed, the long time of training before seeing if the solution worked has made our implementation slower than we thought originally, but the adding of the Batch Normalisation helped us a lot to figure out what was going wrong, because we could test faster new solutions. On a side note, we also tried to implement a L2 regularisation technic by adding it to every Dense layer in our Neural Network, but it did not change enough for us to see results in time. We could focus on the hyperparameters settings of the regularisation to see if it can have a use in improving the accuracy of the decoding.